Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

K-Plane Regression

Naresh Manwani NARESH@EE.IISC.ERNET.IN
Department of Electrical Engineering
Indian Institute of Science
Bangalore 560012, India

P. S. Sastry SASTRY@EE.IISC.ERNET.IN
Department of Electrical Engineering
Indian Institute of Science
Bangalore 560012, India

Editor: Leslie Pack Kaelbling

Abstract

In this paper, we present a novel algorithm for piecewise linear regression which can learn continuous as well as discontinuous piecewise linear functions. The main idea is to repeatedly partition the data and learn a liner model in in each partition. While a simple algorithm incorporating this idea does not work well, an interesting modification results in a good algorithm. The proposed algorithm is similar in spirit to k-means clustering algorithm. We show that our algorithm can also be viewed as an EM algorithm for maximum likelihood estimation of parameters under a reasonable probability model. We empirically demonstrate the effectiveness of our approach by comparing its performance with the state of art regression learning algorithms on some real world datasets.

Keywords: Regression, Mixture Models.

1. Introduction

In a regression problem, given the training dataset containing pairs of multi-dimensional feature vectors and corresponding real-valued target outputs, the task is to learn a function that captures the relationship between feature vectors and their corresponding target outputs.

Least square regression and support vector regression are well known and generic approaches for regression learning problems (Bishop, 2006; Hastie et al., 2001; Smola and Schölkopf, 1998). In the least squares approach, nonlinear regression functions can be learnt by using user-specified fixed nonlinear mapping of feature vectors from original space to some suitable high dimensional space though this could be computationally expensive. In support vector regression (SVR), kernel functions are used for nonlinear problems. Using a nonlinear kernel function, SVR implicitly transforms the examples to some high dimensional space and finds a linear regression function in the high dimensional space. SVR has a large margin flavor and has well studied performance guarantees. In general, SVR solution is not easily interpretable in the original feature space for nonlinear problems.

A different approach to learning a nonlinear regression function is to approximate the target function by a piecewise linear function. Piecewise linear approach for regression

problems provides better understanding of the behavior of the regression surface in the original feature space as compared to the kernel-based approach of SVR. In piecewise linear approaches, the feature space is partitioned into disjoint regions and for every partition a linear regression function is learnt. The goal here is to simultaneously estimate the optimal partitions and linear model for each partition. This problem is hard and is computationally intractable (Paoletti et al., 2007).

The simplest piecewise linear function is either a convex or a concave piecewise linear function which is represented as a maximum or minimum of affine functions. A generic piecewise linear regression function can be represented as a sum of these convex/concave piecewise linear functions (Breiman, 1993; Wang and Sun, 2005; Magnani and Boyd, 2009).

In this paper we present a novel method of learning piecewise linear regression functions. In contrast to all the existing methods, our approach is capable of learning discontinuous functions also. We show, through empirical studies, that this algorithm is attractive in comparison to the SVR approach as well as the hinge hyperplanes method which is among the best algorithms for learning piecewise linear functions.

Existing approaches for piecewise linear regression learning can be broadly classified into two classes. In the first set of approaches one assumes a specific form for the function and estimates the parameters. Form of a regression function can be fixed by fixing the number of hyperplanes and fixing the way these hyperplanes are combined to approximate the regression surface. In the second set of approaches, the form of the regression function is not fixed apriori.

In fixed structure approaches we search over a parameterized family of piecewise linear regression functions and the parameters are learnt by solving an optimization problem to, typically, minimize the sum of the squared errors. Some examples of such methods are mixture of experts and hierarchical mixture of experts (Jacobs et al., 1991; Waterhouse, 1997; Jordan and Jacobs, 1994) models.

In the set of approaches where no fixed structure is assumed, *regression tree* (Breiman et al., 1984; Jayadeva and Chandra, 2002) is the most widely used method. A regression tree is built by binary or multivariate recursive partitioning in a greedy fashion. Regression trees split the feature space at every node in such a way that fitting a linear regression function to each child node will minimize the sum of squared errors. This splitting or partitioning is then applied to each of the child nodes. The process continues until the number of data points at a node reaches a user-specified minimum size or the error becomes smaller than some tolerance limit. In contrast to decision trees where leaf nodes are assigned class labels, leaf nodes in regression trees are associated with linear regression models. Most of the algorithms for learning regression trees are greedy in nature. At any node of the tree, once a hyperplane is learnt to split the feature space, it can not be altered by any of its child nodes. The greedy nature of the method can result in convergence to a suboptimal solution.

A more refined regression tree approach is *hinging hyperplane* method (Breiman, 1993; Pucar and Sjöberg, 1998) which overcomes several drawbacks of regression tree approach. A hinge function is defined as maximum or minimum of two affine functions (Breiman, 1993). In the hinging hyperplane approach, the regression function is approximated as a sum of these hinge functions where the number of hinge functions are not fixed apriori. The algorithm starts with fitting a hinge function on the training data using the hinge finding

algorithm (Breiman, 1993). Then, residual error is calculated for every example and based on this a new hinge function may be added to the model (unless we reach the maximum allowed number of hinges). Every time a new hinge function is added, its parameters are found by fitting the residual error. This algorithm overcomes the greedy nature of regression tree approach by providing a mechanism for re-estimation of parameters of each of the earlier hinge function whenever a new hinge is added. Overall, hinge hyperplanes algorithm tries to learn an optimal regression tree, given the training data.

A different greedy approach for piecewise linear regression learning is *bounded error approach* (Amaldi and Mattavelli, 2002; Bemporad et al., 2003, 2005). In bounded error approaches, for a given bound ($\epsilon > 0$) on the tolerable error, the goal is to learn a piecewise linear regression function such that for every point in the training set, the absolute difference between the target value and the predicted value is less than ϵ. This property is called bounded error property. Greedy heuristic algorithms (Bemporad et al., 2003, 2005) have been proposed to find such a piecewise linear function. These algorithms start with finding a linear regression function which should satisfy the bounded error property for as many points in the training set as possible. This problem is known as *maximum feasible subsystem problem* (MAX-FS) and is shown to be NP-hard (Amaldi and Mattavelli, 2002). MAX-FS problem is repeated on the remaining points until all points are exhausted. So far, there are no theoretical results to support the quality of the solution of these heuristic approaches.

Most of the existing approaches for learning regression functions find a continuous approximation for the regression surface even if the actual surface is discontinuous. In this paper, we present a piecewise linear regression algorithm which is able to learn both continuous as well as discontinuous functions.

We start with a simple algorithm that is similar, in spirit, to the k-means clustering algorithm. The idea is to repeatedly keep partitioning the training data and learning a hyperplane for each partition. In each such iteration, after learning the hyperplanes, we repartition the feature vectors so that all feature vectors in a partition have least prediction error with the hyperplane of that partition. We call it K-plane regression algorithm. Though we are not aware of any literature where such a method is explicitly proposed and investigated for learning regression functions, similar ideas have been proposed in related contexts. For example, a similar problem is addressed in the system identification literature (Amaldi and Mattavelli, 2002). A probabilistic version of such an idea was discussed under the title *mixtures of multiple linear regression* (Bishop, 2006, Chapter 14).

This K-plane regression algorithm is attractive because it is conceptually very simple. However, it suffers from some serious drawbacks in terms of convergence to non-optimal solutions, sensitivity to additive noise and lack of model function. We discuss these issues and based on this insight propose new and modified K-plane regression algorithm. In the modified algorithm also we keep repeatedly partioning the data and learning a linear model for each partition. However, we try to separtely and simultaneously learn the centers of the partions and the corresponding linear models. Through empirical studies we show that this algorithm is very effective for learning piecewise linear regression surfaces and it compares favourably with other state-of-art regression function learning methods.

The rest of the paper is organized as follows. In Section 2 we discuss K-plane regression algorithm, its drawbacks and possible reasons behind them. We then propose modified

K-plane regression algorithm in Section 3. We also show that modified K-plane regression algorithm monotonically decreases the error function after every iteration. In Section 4 we show the equivalence of our algorithm with an EM algorithm in limiting case. Experimental results are given in Section 5. We conclude the paper in Section 6.

2. K-Plane Regression

We begin by defining a K-piecewise affine function. We use the notation that a hyperplane in $\Re^d$ is parameterized by $\tilde{\mathbf{w}} = [\mathbf{w}^T \quad b]^T \in \Re^{d+1}$ where $\mathbf{w} \in \Re^d$ and $b \in \Re$.

Definition 1 *A function $f : \Re^d \to \Re$, is called K-**piecewise affine** if there exists a set of K hyperplanes with parameters $(\mathbf{w}_1, b_1), \ldots, (\mathbf{w}_K, b_K) \in \Re^{d+1}$ $((\mathbf{w}_i, b_i) \neq (\mathbf{w}_j, b_j), i \neq j)$, and sets $\tilde{S}_1, \ldots, \tilde{S}_K \subset \Re^d$ (which form a partition of $\Re^d$), such that, $f(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k, \forall \mathbf{x} \in \tilde{S}_k$.*

From the definition above, it is clear that $(\mathbf{w}_j^T \mathbf{x} + b_j - f(\mathbf{x}))^2 \geq (\mathbf{w}_k^T \mathbf{x} + b_k - f(\mathbf{x}))^2 = 0, \forall \mathbf{x} \in \tilde{S}_k, \forall j \neq k$. Also, note that a K-piecewise affine function may be discontinuous.

K-Plane Regression

Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be the training dataset, where $(\mathbf{x}_n, y_n) \in \Re^d \times \Re$. Let $\tilde{\mathbf{x}}_n = [\mathbf{x}_n^T \quad 1]^T$, $n = 1 \ldots N$. K-plane regression approach tries to find a pre-fixed number of hyperplanes such that each point in the training set is close to one of the hyperplanes. Let K be the number of hyperplanes. Let $\tilde{\mathbf{w}}_k$, $k = 1 \ldots K$, be the parameters of the hyperplanes. K-plane regression minimizes the following error function.

$$E(\Theta) = \sum_{n=1}^{N} \min_{k \in \{1 \ldots K\}} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2$$

where $\Theta = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K\}$. Given the parameters of K hyperplanes, $\tilde{\mathbf{w}}_1 \ldots \tilde{\mathbf{w}}_K$, define sets S_k, $k = 1 \ldots K$, as $S_k := \{\mathbf{x}_n \mid k = \text{argmin}_{j \in \{1, \ldots K\}} (\tilde{\mathbf{w}}_j^T \tilde{\mathbf{x}}_n - y_n)^2\}$ where we break ties by putting $\mathbf{x}_n$ in the set S_k with least k. The sets S_k are disjoint. We can now write $E(\Theta)$ as

$$E(\Theta) = \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2 \tag{1}$$

If we fix all S_k, then $\tilde{\mathbf{w}}_k$ can be found by minimizing (over $\tilde{\mathbf{w}}$) $\sum_{\mathbf{x}_n \in S_k} (\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_n - y_n)^2$. However, in $E(\Theta)$ defined in equation (1), the sets S_k themselves are function of the parameter set $\Theta = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K\}$.

To find Θ which minimize $E(\Theta)$ in (1), we can have an EM-like algorithm as follows. Let, after c^{th} iteration, the parameter set be Θ^c. Keeping Θ^c fixed, we first find sets $S_k^c = \{\mathbf{x}_n \mid k = \text{argmin}_{j \in \{1, \ldots K\}} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^c - y_n)^2\}$, $k = 1 \ldots K$. Now we keep these sets S_k^c, $k = 1 \ldots K$, fixed. Thus the error function becomes

$$E^c(\Theta) \quad = \quad \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2 = \sum_{k=1}^{K} E_k^c(\tilde{\mathbf{w}}_k)$$

Algorithm 1: K-plane regression

Input: $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$
Output: $\{\tilde{\mathbf{w}}_1 \ldots \tilde{\mathbf{w}}_K\}$
begin
Step 1: Initialize $\tilde{\mathbf{w}}_k^0, k = 1 \ldots K$, Initialize $c = 0$.
Step 2: Find sets $S_k^c, k = 1 \ldots K$

$$S_k^c = \{\mathbf{x}_n \mid k = \text{argmin}_{j \in \{1 \ldots K\}} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^c - y_n)^2\}$$

Step 3: Find $\mathbf{w}_k^{c+1}, k = 1 \ldots K$, as follows

$$\tilde{\mathbf{w}}_k^{c+1} = \Big[\sum_{\mathbf{x}_n \in S_k^c} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T\Big]^{-1} \big[\sum_{\mathbf{x}_n \in S_j^c} y_n \tilde{\mathbf{x}}_n\big]$$

Step 4: Find sets $S_k^{c+1}, k = 1 \ldots K$

$$S_k^{c+1} = \{\mathbf{x}_n \mid k = \text{argmin}_{j \in \{1 \ldots K\}} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^{c+1} - y_n)^2\}$$

Step 5: Termination Criteria
if $S_k^{c+1} = S_k^c, k = 1 \ldots K$ **then**
stop
else
$c = c + 1$
go to *Step 3*
end
end

where superscript c denotes the iteration and hence emphasizes the fact that the error function is evaluated by fixing the sets $S_k^c, k = 1 \ldots K$, and

$$E_k^c(\tilde{\mathbf{w}}_k) = \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2. \tag{2}$$

Thus, minimizing $E^c(\Theta)$ with respect to Θ boils down to minimizing each of $E_k^c(\tilde{\mathbf{w}}_k)$ with respect to $\tilde{\mathbf{w}}_k$. For every $k \in \{1, \ldots, K\}$, a new weight vector $\tilde{\mathbf{w}}_k^{c+1}$ is found using standard linear least square solution as follows.

$$\tilde{\mathbf{w}}_k^{c+1} \;=\; \text{argmin}_{\tilde{\mathbf{w}}_k} \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2 = \Big[\sum_{\mathbf{x}_n \in S_k^c} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T\Big]^{-1} \big[\sum_{\mathbf{x}_n \in S_k^c} y_n \tilde{\mathbf{x}}_n\big] \tag{3}$$

Now we fix Θ^{c+1} and find new sets $S_k^{c+1}, k = 1 \ldots K$, and so on. We can now summarize K-plane regression algorithm. We first find sets $S_k^c,\ k = 1 \ldots K$, for iteration c (using $\tilde{\mathbf{w}}_k^c,\ k = 1 \ldots K$). Then for each $k = 1 \ldots K$, we find $\tilde{\mathbf{w}}_k^{c+1}$ (as in equation (3)) by minimizing $E_k^c(\tilde{\mathbf{w}}_k)$ which is defined in equation (2). We keep on repeating these two steps until there is no significant decrement in the error function $E(\Theta)$. $E(\Theta)$ does not change when the weight vectors do not change or sets $S_k,\ k = 1 \ldots K$, do not change. The complete K-plane regression approach is described more formally in Algorithm 1.

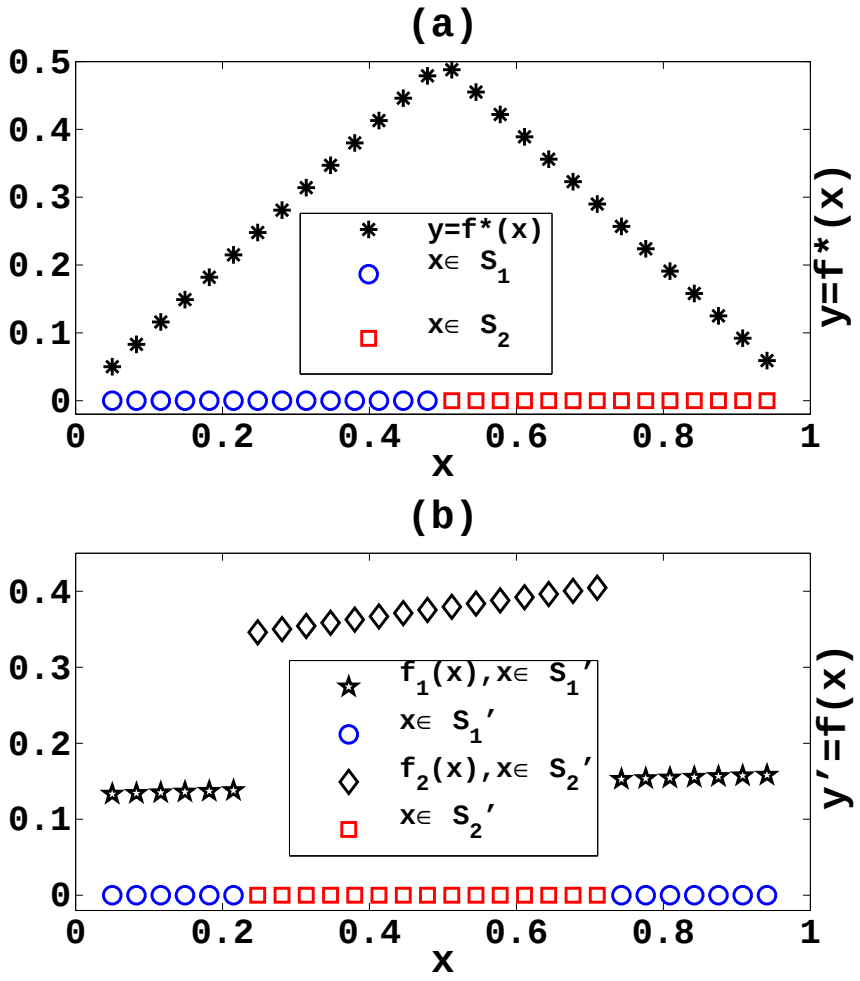

Figure 1: (a) Points sampled from a triangle shaped function $f^*(x)$, (b) function $f(x)$ learnt using K-plane regression algorithm given the points sampled from function $f^*(x)$.

2.1 Issues with K-Plane Regression

In spite of its simplicity and easy updates, K-plane regression algorithm has some serious drawbacks in terms of convergence and model issues.

1. Convergence to Non-optimal Solutions

It is observed that the algorithm has serious problem of convergence to non-optimal solution. Even when the data is generated from a piecewise linear function, the algorithm often fails to learn the structure of the target function.

Figure 1(a) shows points sampled from a concave (triangle shaped) 2-piecewise affine function on the real line. At the horizontal axis, circles represent set S_1 and squares represent set S_2, where S_1 and S_2 constitute the correct partitioning of the training set in this problem.

We see that convex hulls of sets S_1 and S_2 are disjoint. This 2-piecewise affine function can be written (as per defn. 1) by choosing $\tilde{S}_1,\ \tilde{S}_2$ to be the convex hulls of S_1 and S_2.

Figure 1(b) shows the 2-piecewise linear function learnt using K-plane regression approach for a particular initialization. S'_1 (represented as circles) and S'_2 (represented as squares) are sets corresponding to the two lines in the learnt function. Here, K-plane regression algorithm completely misses the shape of the target function. We also see that convex hulls of sets S'_1 and S'_2 intersect with each other.

2. Sensitivity to Noise

It has been observed in practice that the simple K-plane regression algorithm is very sensitive to the additive noise in the target values in training set. Under noisy examples, the algorithm performs badly. We illustrate it later in Section 5.

3. Lack of Model Function

The output of the K-plane regression algorithm is a set of K hyperplanes. But this algorithm does not provide a way to use these hyperplanes to predict the value for a given test point. In other words, K-plane regression algorithm does not have any model function for prediction. We expand this issue in the next section.

3. Modified K-Plane Regression

As we have mentioned, given the training data, $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, the K-plane regression algorithm outputs K hyperplanes, $\tilde{\mathbf{w}}_k^*,\ k = 1 \ldots K$. To convert this into a proper K-piecewise linear model in $\Re^d$, we also need to have a K-partition of $\Re^d$ such that in the k^{th} partition, the appropriate model to use would be $\tilde{\mathbf{w}}_k^*$. We could attempt to get such a partition of $\Re^d$ by considering the convex hulls of $S_k^*,\ k = 1 \ldots K$ (where $S_k^* = \{\mathbf{x}_n \mid k = \text{argmin}_j (y_n - \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^*)^2\}$). However, as we saw, under the K-plane regression, the convex hulls of such S_k^* need not be disjoint. Hence another method to get the required partition is as follows. Let $\boldsymbol{\mu}_k^*$ be the mean or centroid of S_k^*. Then, for any point, $\mathbf{x} \in \Re^d$, our prediction could be $\hat{y} = \mathbf{x}^T \tilde{\mathbf{w}}_j^*$, where j is such that $||\mathbf{x} - \boldsymbol{\mu}_j^*|| \leq ||\mathbf{x} - \boldsymbol{\mu}_k^*||,\ \forall k \neq j$ (break ties arbitrarily). This would define a proper model function with the hyperplanes obtained through K-plane regression. However, this may not give good performance. Often, the convex hulls of sets, $S_k^*,\ k = 1 \ldots K$ (learnt using K-plane regression), have non-null intersection because each of these sets may contain points from different disjoint regions of $\Re^d$ (for example, see Figure 1). In such cases, if we re-partition the training data using distances to different $\boldsymbol{\mu}_j^*$, we may get sets much different from S_k^* and hence our final prediction on even training data may have large error. The main reason for this problem with K-plane regression is that the algorithm is not really bothered about the geometry of the sets S_k; it only focuses on $\tilde{\mathbf{w}}_k$ to be a good fit for points in set S_k. Moreover, in situations where same affine function works for two or more disjoint clusters, k-plane regression will consider them as a single cluster as the objective function of k-plane regression does not enforce that points in the same cluster should be close to each other. As a result, the clusters learnt using K-plane regression approach will have overlapping convex hulls and some times even their means may be very close to each other. This may create problems during prediction.

If we use the hyperplane whose corresponding cluster mean is closest to a point, then we may not pick up the correct hyperplane. This identifiability problem of K-plane regression approach results in poor performance.

Motivated by this, we modify K-plane regression as follows. We want to simultaneously estimate $\tilde{\mathbf{w}}_k,\ k = 1 \dots K$ and $\boldsymbol{\mu}_k,\ k = 1 \dots K$, such that if, $\tilde{\mathbf{w}}_k$ is a good fit for k^{th} partition, all the points in k^{th} partition should be closer to $\boldsymbol{\mu}_k$ than any other $\boldsymbol{\mu}_j^*$. Intuitively, we can think of $\boldsymbol{\mu}_k$ as center of the (cluster or) set of points S_k. However, as we saw from our earlier example, if we simply make $\boldsymbol{\mu}_k$ as the centroid of the final S_k learnt, all the earlier problem still remain. Hence, in the modified K-plane regression, we try to independently learn both $\tilde{\mathbf{w}}_k$ and $\boldsymbol{\mu}_k$ from the data. To do that, we add an extra term to the objective function of K-plane regression approach which tries to ensure that all the points of same cluster are close together.

As earlier, let the number of hyperplanes be K. Here, in the modified K-plane regression, we have to learn $2K$ parameter vectors. Corresponding to k^{th} partition, we have two parameter vectors, $\tilde{\mathbf{w}}_k \in \Re^{d+1}$ and $\boldsymbol{\mu}_k \in \Re^d$. $\tilde{\mathbf{w}}_k$ represents parameter vector of the hyperplane associated with the k^{th} partition and $\boldsymbol{\mu}_k$ represents center of the k^{th} partition. Note that we want to simultaneously learn both $\tilde{\mathbf{w}}_k$ and $\boldsymbol{\mu}_k$ for every partition.

The error function minimized by modified K-plane regression algorithm is

$$E(\Theta) = \sum_{n=1}^{N} \min_{k \in \{1,\dots,K\}} \left[(\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2 \right] \tag{4}$$

where $\Theta = \{(\tilde{\mathbf{w}}_1, \boldsymbol{\mu}_1), \dots, (\tilde{\mathbf{w}}_K, \boldsymbol{\mu}_K)\}$ and γ is a user defined parameter which decides relative weight of the two terms.

Given Θ, we define sets $S_k,\ k = 1 \dots K$, as

$$S_k := \left\{ \mathbf{x}_n \mid k = \text{argmin}_{j \in \{1,\dots K\}} \left[(\tilde{\mathbf{w}}_j^T \tilde{\mathbf{x}}_n - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j||^2 \right] \right\}$$

where we break ties by putting $\mathbf{x}_n$ in the set S_k with least k. The sets S_k are disjoint. We can now write $E(\Theta)$ as

$$E(\Theta) = \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2 \tag{5}$$

As can be seen from the above, now, for a data point, $\mathbf{x}_n$ to be put in S_k, we not only need $\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n$ to be close to y_n as earlier, but also need $\mathbf{x}_n$ to be close to $\boldsymbol{\mu}_k$, the 'current center' of S_k. The motivation is that, under such a partitioning strategy, each S_k would contain only points that are close to each other. As we shall see later through simulations, this modification ensures that the algorithm performs well. As an example of where this modification is important, consider learning a piecewise linear model which is given by same affine function in two (or more) disjoint regions in the feature space. For any splitting of all examples from these two regions into two parts, there will be a good linear model that fits each of the two parts. Hence, in the K-plane regression method, the $E(\Theta)$ function (cf.eq.(1)) would be same for any splitting of the examples from these two regions which means we would not learn a good model. However, the modified K-plane regression

approach will not treat all such splits as same because of the term involving $\boldsymbol{\mu}_k$. This helps us learn a proper piecewise linear regression function. We illustrate this in Section 5.

Now consider finding Θ to minimize $E(\Theta)$ given by equation (5). If we fix all S_k, then $\tilde{\mathbf{w}}_k$ and $\boldsymbol{\mu}_k$ can be found by minimizing (over $\tilde{\mathbf{w}},\ \boldsymbol{\mu}$) $\sum_{\mathbf{x}_n \in S_k} (\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_n - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}||^2$. However, in $E(\Theta)$ defined in equation (5), the sets S_k themselves are functions of the parameter set $\Theta = \{(\tilde{\mathbf{w}}_1, \boldsymbol{\mu}_1), \ldots, (\tilde{\mathbf{w}}_K, \boldsymbol{\mu}_K)\}$.

To find Θ which minimize $E(\Theta)$ in (5), we can, once again, have an EM-like algorithm as follows. As earlier, let the parameter set after c^{th} iteration be Θ^c. Keeping Θ^c fixed, we find the sets $S_k^c,\ k = 1 \ldots K$, as follows

$$S_k^c = \left\{ \mathbf{x}_n \mid k = \text{argmin}_{j \in \{1,\ldots K\}} [(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^c - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j^c||^2] \right\} \tag{6}$$

Now we keep these sets S_k^c fixed. Thus the error function becomes

$$E^c(\Theta) = \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k^c} [(\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2] = \sum_{k=1}^{K} E^c(\tilde{\mathbf{w}}_k, \boldsymbol{\mu}_k)$$

where superscript c denotes the iteration and emphasizes the fact that the error function is evaluated by fixing the sets $S_k^c,\ k = 1 \ldots K$. Thus minimizing $E^c(\Theta)$ with respect to Θ boils down to minimizing each of $E_k^c(\tilde{\mathbf{w}}_k, \boldsymbol{\mu}_k)$ with respect to $(\tilde{\mathbf{w}}_k, \boldsymbol{\mu}_k)$. Each $E_k^c(\tilde{\mathbf{w}}_k, \boldsymbol{\mu}_k)$ is composed of two terms. The first term depends only on $\tilde{\mathbf{w}}_k$ and and it is the usual sum of squares of errors. The second term depends only on $\boldsymbol{\mu}_k$ and it is the usual cost function of K-means clustering. Thus, the update equations for finding $\tilde{\mathbf{w}}_k^{c+1}$ and $\boldsymbol{\mu}_k^{c+1}$, $k = 1 \ldots K$, are

$$\begin{aligned}
\tilde{\mathbf{w}}_k^{c+1} &= \text{argmin}_{\tilde{\mathbf{w}}_k} \sum_{j=1}^{K} E^c(\tilde{\mathbf{w}}_j, \boldsymbol{\mu}_j) = \text{argmin}_{\tilde{\mathbf{w}}} \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_n - y_n)^2 \\
&= \Big[\sum_{\mathbf{x}_n \in S_k^c} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T \Big]^{-1} \Big[\sum_{\mathbf{x}_n \in S_k^c} y_n \tilde{\mathbf{x}}_n \Big] \qquad (7) \\
\boldsymbol{\mu}_k^{c+1} &= \text{argmin}_{\boldsymbol{\mu}_k} \sum_{j=1}^{K} E^c(\tilde{\mathbf{w}}_j, \boldsymbol{\mu}_j) = \text{argmin}_{\boldsymbol{\mu}} \sum_{\mathbf{x}_n \in S_k^c} ||\mathbf{x}_n - \boldsymbol{\mu}||^2 \\
&= \frac{1}{|S_k^c|} \sum_{\mathbf{x}_n \in S_k^c} \mathbf{x}_n \qquad (8)
\end{aligned}$$

Once we compute Θ^{c+1}, we find new sets $S_k^{c+1}, k = 1 \ldots K$, and so on.

In summary, the modified K-plane regression algorithm works as follows. We first find sets $S_k^c,\ k = 1 \ldots K$, for iteration c (using $(\tilde{\mathbf{w}}_k^c, \boldsymbol{\mu}_k^c),\ k = 1 \ldots K$) as given by eq.(6). Then for each $k = 1 \ldots K$, we find $(\tilde{\mathbf{w}}_k^{c+1}, \boldsymbol{\mu}_k^{c+1})$ (as in equation (7) and (8)) by maximizing $E_k^c(\tilde{\mathbf{w}}_k, \boldsymbol{\mu}_k)$. We keep on repeating these two steps until there is no significant decrement in the error function $E(\Theta)$. The complete description of modified K-plane regression approach is given in Algorithm 2.

Algorithm 2: Modified K-plane regression

Input: $\{(\mathbf{x}_1, y_1) \dots (\mathbf{x}_N, y_N)\}$
Output: $\{(\tilde{\mathbf{w}}_1, \boldsymbol{\mu}_1) \dots (\tilde{\mathbf{w}}_K, \boldsymbol{\mu}_K)\}$
begin
Step 1: Initialize $(\mathbf{w}_k^0, \boldsymbol{\mu}_k^0),\ k = 1 \dots K$. Initialize $c = 0$.
Step 2: Find $S_k^c, k = 1 \dots K$, as follows

$$S_k^c = \left\{\mathbf{x}_n \mid k = \text{argmin}_{j \in \{1 \dots K\}} [(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^c - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j^c||^2]\right\}$$

Step 3: Find $\tilde{\mathbf{w}}_k^{c+1}, \boldsymbol{\mu}_k^{c+1}, k = 1 \dots K$, as follows

$$\begin{aligned} \tilde{\mathbf{w}}_k^{c+1} &= \Big[\sum_{\mathbf{x}_n \in S_k^c} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T\Big]^{-1} \Big[\sum_{\mathbf{x}_n \in S_k^c} y_n \tilde{\mathbf{x}}_n\Big] \\ \boldsymbol{\mu}_k^{c+1} &= \frac{1}{|S_k^c|} \sum_{\mathbf{x}_n \in S_k^c} \mathbf{x}_n \end{aligned}$$

Step 4: Find $S_k^{c+1}, k = 1 \dots K$, as follows

$$S_k^{c+1} = \left\{\mathbf{x}_n \mid k = \text{argmin}_{j \in \{1 \dots K\}} [(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j^{c+1}||^2]\right\}$$

Step 5: Termination Criteria
if $S_k^{c+1} = S_k^c, \forall k$ **then**
stop
else
$c = c + 1$
go to *Step 3*
end
end

Monotone Error Decrement Property

Now we will show that modified K-plane regression algorithm monotonically decreases the error function defined by equation (4).[1]

Theorem 1 *Algorithm 2 monotonically decreases the cost function given by equation (4) after every iteration.*

Proof We have

$$E^c(\Theta^c) = \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k^c} \left[(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^c - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k^c||^2\right]$$

1. Note that this does not necessarily mean that we find the global minimum of the error function. More importantly, we can not claim that minimizing the error as defined would lead to learning of a good piecewise linear model. We note here that the simple K-plane regression algorithm also results in monotonic decrease in the error as defined for that algorithm even though it may not learn good models. However, the fact that the algorithm continuously decreases the error at each iteration, is an important property for a learning algorithm.

Given the sets $S_k^c,\ k = 1 \dots K$, parameters $(\tilde{\mathbf{w}}_k^{c+1}, \boldsymbol{\mu}_k^{c+1}),\ k = 1 \dots K$, are found using equation (7) and (8), in the following way.

$$\begin{aligned} \tilde{\mathbf{w}}_k^{c+1} &= \text{argmin}_{\tilde{\mathbf{w}}_k} \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2 \\ \boldsymbol{\mu}_k^{c+1} &= \text{argmin}_{\boldsymbol{\mu}_k} \sum_{\mathbf{x}_n \in S_k^c} ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2 \end{aligned}$$

Thus, we have

$$\begin{aligned} \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^c - y_n)^2 \geq \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^{c+1} - y_n)^2,\ k = 1 \dots K \\ \sum_{\mathbf{x}_n \in S_k^c} ||\mathbf{x}_n - \boldsymbol{\mu}_k^c||^2 \geq \sum_{\mathbf{x}_n \in S_k^c} ||\mathbf{x}_n - \boldsymbol{\mu}_k^{c+1}||^2,\ k = 1 \dots K \end{aligned}$$

This will further give us

$$\sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^c - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k^c||^2 \geq \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k^{c+1}||^2,\ \forall k$$

$$\begin{aligned} &\Rightarrow \sum_{k=1}^K \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^c - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k^c||^2 \geq \sum_{k=1}^K \sum_{\mathbf{x}_n \in S_k^c} (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k^{c+1}||^2 \\ &\Rightarrow E^c(\Theta^c) \geq E^c(\Theta^{c+1}) \end{aligned} \tag{9}$$

Given Θ^{c+1}, sets $S_k^{c+1}, k = 1 \dots K$, are found as follows

$$S_k^{c+1} = \left\{ \mathbf{x}_n \mid k = \text{argmin}_{j \in \{1 \dots K\}} [(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j^{c+1}||^2] \right\}$$

Using $S_k^{c+1},\ k = 1 \dots K$, we can find $E^{c+1}(\Theta^{c+1})$, which is

$$\begin{aligned} E^{c+1}(\Theta^{c+1}) &= \sum_{k=1}^K \sum_{\mathbf{x}_n \in S_k^{c+1}} \left[(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k^{c+1}||^2 \right] \\ &= \sum_{k=1}^K \sum_{j=1}^K \sum_{\mathbf{x}_n \in S_j^c \cap S_k^{c+1}} \left[(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k^{c+1}||^2 \right] \end{aligned}$$

By the definition of sets S_k^{c+1}, $\forall \mathbf{x}_n \in S_k^{c+1}$, we have,

$$(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k^{c+1}||^2 \leq (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j^{c+1}||^2,\ \forall j \neq k$$

which is also true for any $\mathbf{x}_n \in S_k^{c+1} \cap S_j^c$. Thus

$$\begin{aligned} E^{c+1}(\Theta^{c+1}) &\leq \sum_{k=1}^K \sum_{j=1}^K \sum_{\mathbf{x}_n \in S_j^c \cap S_k^{c+1}} \left[(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j^{c+1}||^2 \right] \\ &= \sum_{j=1}^K \sum_{\mathbf{x}_n \in S_j^c} \left[(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^{c+1} - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j^{c+1}||^2 \right] = E^c(\Theta^{c+1}) \end{aligned} \tag{10}$$

Combining (9) and (10), we get $E^c(\Theta^c) \geq E^c(\Theta^{c+1}) \geq E^{c+1}(\Theta^{c+1})$. Which means after one complete iteration modified K-plane regression Algorithm decreases the error function. ■

4. EM View of Modified K-Plane Regression Algorithm

Here, we show that modified K-plane regression algorithm presented in Section 3 can be viewed as a limiting case of an EM algorithm. In the general K-plane regression idea, the difficulty is due to the following credit assignment problem. When we decompose the problem into K subproblems, we do not know which $\mathbf{x}_n$ should be considered in which subproblem. We can view this as the missing information in the EM formulation.

Recall that $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ is the training data set. In the EM framework, $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ can be thought of as incomplete data. The missing data would be $\mathbf{z}_n = [z_{n1} \ \ldots \ z_{nK}]^T$, where $z_{nk} \in \{0, 1\}, \ \forall n, \ \forall k$, such that, $\sum_{k=1}^K z_{nk} = 1, \ \forall n$. Then z_{nk} are defined as,

$$z_{nk} = \begin{cases} 1, & \text{if } (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2 \leq (\tilde{\mathbf{w}}_j^T \tilde{\mathbf{x}}_n - y_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_j||^2, \ \forall j \neq k \\ 0, & \text{otherwise} \end{cases}$$

This gives us the following probability model

$$\begin{cases} P(\mathbf{x}_n, y_n | z_{nk} = 1, \Theta) & = p(\mathbf{x}_n, y_n | \tilde{\mathbf{w}}_k, \boldsymbol{\mu}_k) = p(\mathbf{x}_n | \boldsymbol{\mu}_k) p(y_n | \mathbf{x}_n, \tilde{\mathbf{w}}_k) \\ P(\mathbf{x}_n, y_n | \mathbf{z}_n, \Theta) & = \sum_{k=1}^K z_{nk} p(\mathbf{x}_n | \boldsymbol{\mu}_k) p(y_n | \mathbf{x}_n, \tilde{\mathbf{w}}_k) \end{cases} \tag{11}$$

In our formulation, $\boldsymbol{\mu}_k$ represents the center of the set of all $\mathbf{x}$ for which the k^{th} linear model is appropriate. Hence we take $p(\mathbf{x}|\boldsymbol{\mu}_k) = \mathcal{N}(\boldsymbol{\mu}_k, \frac{\epsilon}{\gamma} I)$, a multivariate Gaussian in which the covariance matrix is given by $\frac{\epsilon}{\gamma} I$, where $\frac{\epsilon}{\gamma} (\epsilon, \gamma > 0)$ is a variance parameter, and I is the identity matrix. This covariance matrix is common for all K components. We assume that the target values given in the training set may be corrupted with zero mean Gaussian noise. Thus, for k^{th} component, the target value is assigned using $\tilde{\mathbf{w}}_k$ as $y = \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}} + e$, where e is Gaussian noise with mean 0 and variance ϵ. Variance ϵ is kept same for all K components. Thus, $p(y|\mathbf{x}, \tilde{\mathbf{w}}_k) = \mathcal{N}(\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n, \epsilon)$, a Gaussian with mean $\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n$ and variance ϵ. Thus,

$$\begin{aligned} p(\mathbf{x}_n, y_n | \tilde{\mathbf{w}}_k, \boldsymbol{\mu}_k) &= \mathcal{N}(\boldsymbol{\mu}_k, \frac{\epsilon}{\gamma} I) \mathcal{N}(\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n, \epsilon) \\ &= \frac{\gamma^{\frac{d}{2}}}{(2\pi\epsilon)^{\frac{d}{2}}} \exp\big(-\frac{\gamma}{2\epsilon} ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2\big) \frac{1}{(2\pi\epsilon)^{\frac{1}{2}}} \exp\big(-\frac{1}{2\epsilon} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n - y_n)^2\big) \\ &= \frac{1}{L} \exp\big(-\frac{1}{2\epsilon} [(y_n - \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2]\big) \end{aligned}$$

where $L = \left[\frac{(2\pi\epsilon)^{(d+1)}}{\gamma^d}\right]^{\frac{1}{2}}$. Note that ϵ and γ are assumed to be fixed constant, instead of parameters to be re-estimated. Thus, the density model for incomplete data becomes

$$\begin{aligned} p(\mathbf{x}_n, y_n | \Theta) &= \sum_{k=1}^K \mathbf{I}_{\{k=a_n\}} \frac{1}{L} \exp\Big(-\frac{1}{2\epsilon} [(y_n - \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2]\Big) \\ &= \frac{1}{L} \exp\Big(-\frac{1}{2\epsilon} \min_k [(y_n - \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n)^2 + \gamma ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2]\Big) \end{aligned} \tag{12}$$

Negative of the log-likelihood under the model given in equation (12) is same as the error function minimized in the modified K-plane regression algorithm. Hence, we can now compute the EM iteration for maximizing log-likelihood computed from (11).

However, the incomplete data log-likelihood under our probability model (12) becomes non-differentiable due to the hard minimum function. To get around this, we change the probability model for incomplete data into a mixture model with mixing coefficients as part of Θ:

$$p(\mathbf{x}_n, y_n|\Theta) = \frac{1}{L}\sum_{k=1}^{K} \alpha_k \exp\big(-\frac{1}{2\epsilon}[(y_n - \tilde{\mathbf{w}}_k^T\tilde{\mathbf{x}}_n)^2 + \gamma||\mathbf{x}_n - \boldsymbol{\mu}_k||^2]\big) \tag{13}$$

where $\alpha_k = P(z_{nk} = 1),\ \forall n; \alpha_k \geq 0,\ \sum_{k=1}^{K}\alpha_k = 1$, and $\Theta = \{(\alpha_1, \tilde{\mathbf{w}}_1, \boldsymbol{\mu}_1), \ldots, (\alpha_K, \tilde{\mathbf{w}}_K, \boldsymbol{\mu}_K)\}$. Note that here $p(y_n|\mathbf{x}_n, \Theta) \propto \sum_{k=1}^{K} \frac{\alpha_k \exp(-\frac{\gamma}{2\epsilon}||\mathbf{x}_n - \boldsymbol{\mu}_k||^2)}{\sum_{j=1}^{K}\alpha_j \exp(-\frac{\gamma}{2\epsilon}||\mathbf{x}_n - \boldsymbol{\mu}_j||^2)} \exp(-\frac{1}{2\epsilon}(y_n - \tilde{\mathbf{w}}_k^T\tilde{\mathbf{x}}_n)^2)$, which is same as the model described in Xu et al. (1995) for a mixture of experts network. The incomplete data log-likelihood given by (13) will now be smooth and we can use EM algorithm to maximize the likelihood. However, the model given in (13) is somewhat different from the one in equation (12) which was used in Section 3.

We, now derive the iterative scheme under EM framework using the model specified by equation (11) and (13) and show that in the limit $\epsilon \to 0$, the iterative scheme becomes the modified K-plane regression algorithm that we presented in Section 3.

4.1 EM Algorithm

We now describe EM algorithm with $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ as incomplete data and $\bar{S} = \{(\mathbf{x}_1, y_1, \mathbf{z}_1), \ldots, (\mathbf{x}_N, y_N, \mathbf{z}_N)\}$ as complete data and under the model specified by (11) and (13). The complete data log-likelihood is

$$\begin{aligned} l_{\text{complete}}(\Theta; \bar{S}) &= \ln\big[\prod_{n=1}^{N}\prod_{k=1}^{K}[P(\mathbf{x}_n, y_n, z_{nk}|\Theta)]^{z_{nk}}\big] = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}\ln\big[P(z_{nk})P(\mathbf{x}_n, y_n|z_{nk}, \Theta)\big] \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}\Big[\ln\alpha_k - \ln L - \frac{(y - \tilde{\mathbf{w}}_k^T\tilde{\mathbf{x}}_n)^2}{2\epsilon} - \frac{\gamma||\mathbf{x}_n - \boldsymbol{\mu}_k||^2}{2\epsilon}\Big] \end{aligned}$$

E-Step: In E-Step, we find $Q(\Theta, \Theta^c)$ which is the expectation of complete data log-likelihood.

$$\begin{aligned} \mathcal{Q}(\Theta, \Theta^c) &= \mathbb{E}_{\{\mathbf{z}_1,\ldots,\mathbf{z}_N\}}\big[l_{\text{complete}}(\Theta; \bar{S})|\Theta^c\big] \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K}[\ln\alpha_k - \ln L - \frac{(y - \tilde{\mathbf{w}}_k^T\tilde{\mathbf{x}}_n)^2}{2\epsilon} - \frac{\gamma||\mathbf{x}_n - \boldsymbol{\mu}_k||^2}{2\epsilon}]P(z_{nk} = 1|\mathbf{x}_n, y_n, \Theta^c) \end{aligned}$$

M-Step: In the M-Step, we maximize $\mathcal{Q}(\Theta, \Theta^c)$ with respect to Θ to find out new parameter set Θ^{c+1}. This will give us following update equations.

$$\begin{aligned}
\alpha_k^{c+1} &= \frac{1}{N}\sum_{n=1}^{N} P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c) \\
\tilde{\mathbf{w}}_k^{c+1} &= \big[\sum_{n=1}^{N} P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c)\tilde{\mathbf{x}}_n\tilde{\mathbf{x}}_n^T\big]^{-1}\big[\sum_{n=1}^{N} P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c)y_n\tilde{\mathbf{x}}_n\big] \\
\boldsymbol{\mu}_k^{c+1} &= \frac{\sum_{n=1}^{N} P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c)\mathbf{x}_n}{\sum_{n=1}^{N} P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c)}
\end{aligned}$$

where $P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c)$ is given by

$$\begin{aligned}
P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c) &= \frac{\alpha_k^c p(\mathbf{x}_n, y_n|z_{nk}=1, \Theta^c)}{\sum_{j=1}^{K}\alpha_j^c p(\mathbf{x}_n, y_n|z_{nk}=1, \Theta^c)} \\
&= \frac{\alpha_k^c \exp\big(-\frac{1}{2\epsilon}[(y_n - \tilde{\mathbf{x}}_n^T\tilde{\mathbf{w}}_k^c)^2 + \gamma||\mathbf{x}_n - \boldsymbol{\mu}_k^c||^2]\big)}{\sum_{j=1}^{K}\alpha_j^c \exp\big(-\frac{1}{2\epsilon}[(y_n - \tilde{\mathbf{x}}_n^T\tilde{\mathbf{w}}_j^c)^2 + \gamma||\mathbf{x}_n - \boldsymbol{\mu}_j^c||^2]\big)}
\end{aligned}$$

4.2 Limiting Case ($\epsilon \to 0$)

Now consider $\lim_{\epsilon\to 0} P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c)$. Let $a_n^c = \text{argmin}_{j\in\{1,\ldots K\}}\big[(y_n - \tilde{\mathbf{x}}_n^T\tilde{\mathbf{w}}_j^c)^2 + \gamma||\mathbf{x}_n - \boldsymbol{\mu}_k^c||^2\big]$. When $\epsilon \to 0$, then in the denominator, the term corresponding to index a_n^c will go to zero most slowly and hence $\lim_{\epsilon\to 0} P(z_{nk}=1|\mathbf{x}_n, y_n, \Theta^c) = \mathbf{I}_{\{k=a_n^c\}}$, where $\mathbf{I}_{\{k=a_n^c\}} = 1$ if $k = a_n^c$ and zero otherwise. In this limiting case, the EM updates of $\tilde{\mathbf{w}}_k$ and $\boldsymbol{\mu}_k$ will be same as updates of modified K-plane regression algorithm.

5. Experiments

In this section we present empirical results to show the effectiveness of modified K-plane regression approach. We demonstrate how the learnt functions differ among various regression approaches using two synthetic problems. We test the performance of our algorithm on several real datasets also. We compare our approach with hinging hyperplane algorithm which is the best state-of-art regression tree algorithm and with support vector regression (SVR) which is among the best generic regression approaches today.

Dataset Description

The two synthetic datasets are generated as follows:

1. **Problem 1:** In this, points are uniformly sampled from the interval [0 5]. Then, for every point x the target values y are assigned as $y = f(x) + \epsilon$, where

$$f(x) = \begin{cases} x, & \text{if } 0 \le x < 1 \\ 2 - x, & \text{if } 1 \le x < 2 \\ \frac{1}{3}(7 - 2x), & \text{if } 2 \le x < 3.5 \\ \frac{1}{3}(2x - 7), & \text{if } 3.5 \le x \le 5 \end{cases}$$

and ϵ is a Gaussian random variable with zero mean and variance 0.01. 500 points are generated for training and 500 points are generated for testing.

2. **Problem 2:** Points are uniformly sampled from the interval $[0 \quad 3]$. Then, for every point x the target values y are assigned as $y = f(x)$, where

$$f(x) = \begin{cases} x, & \text{if } 0 \leq x < 1 \\ 1, & \text{if } 1 \leq x < 2 \\ x, & \text{if } 2 \leq x \leq 3 \end{cases}$$

We also generate y' as $y' = f(x) + \epsilon$, where ϵ is a Gaussian random variable with zero mean and variance 0.01. 300 points are generated for training and 300 points are generated for testing.

Note that both the above functions are discontinuous.

We also present the experimental comparisons on 4 'real' datasets downloaded from UCI ML repository (A. Asuncion, 2007) which are described in Table 1. In our simulations, we scale all feature values to the range of $[-1 \quad 1]$.

Data set	Dimension	# Points
Boston Housing	13	506
Abalone	8	4177
Auto-mpg	7	392
Computer activity	12	8192

Table 1: Details of real world datasets used from UCI ML repository.

Experimental Setup

We implemented K-plane regression and modified K-plane regression algorithms in MATLAB.[2] We have also implemented hinging hyperplane method in MATLAB. For support vector regression, we used Libsvm (Chang and Lin, 2001) code. All the simulations are done on a PC (Core2duo, 2.3GHz, 2GB RAM).

Modified K-plane regression has one user defined parameter which is γ. We search for the best value of γ using 10-fold cross validation and use that value in our simulations. Both K-plane regression and modified K-plane regression approaches require K (number of hyperplanes) to be fixed apriori. In our experiments, we change the value of K form 2 to 5. Similarly, in hinging hyperplane method, maximum number of hinge functions should be specified. In our simulations, this number is varied from 1 to 5. Support vector regression has three user defined parameters: penalty parameter C, width parameter σ for Gaussian kernel and tolerance parameter ϵ. Best values for these parameters are found using 10-fold cross-validation and the results reported are with these parameters.

2. For K-plane regression, there is no specified model function which can be used to predict the value for a test point. In our simulations, to assign value for any test point using K-plane regression, we use the same methodology as modified K-plane regression approach. That is, using the $\tilde{\mathbf{w}}_k$ learnt, we obtain sets S_k as explained in Section 2; then we find the k such that centroid of S_k is closest to the test point and then use that $\tilde{\mathbf{w}}_k$ to predict the target.

Method	Parameters	MSE
K-plane	# hyperplanes = 4	0.0557
Modified K-plane	# hyperplanes = 4	0.037
Hinge Hyperplane	# hinges = 6	0.0451
SVR	$C = 64$, $\sigma = 16$, $\epsilon = 2^{-5}$	0.013

Table 2: MSE of different regression approaches on problem 1.

Simulation Results: Synthetic Problems

Problem 1: Figure 2 shows functions learnt using different approaches on problem 1 and Table 2 shows MSE achieved with different approaches on a test set. Hinge hyperplane approach and support vector regression (SVR) methods give continuous approximations to the function i f (see Figs. 2(c) and 2(d)). While SVR gets the shape of the function well, the function learnt using SVR is not piecewise linear. Figure 2(e) shows 4-piecewise affine function learnt using K-plane regression method. We see that the K-plane regression approach completely misses the shape of the function which results in a very high MSE. In contrast, as can be seen from figure 2(f), modified K-plane regression approach learns the discontinuous function f exactly (eventhough the function values given in training set are noisy).

Recall that in modified K-plane regression, we essentially partition the data and learn a hyperplane as well as a 'center' or 'mean' (which was called $\boldsymbol{\mu}_k$ in the algorithm) for each partition. The target function in this example has four linear pieces. If we got the exact partitioning of the input space then the ideal centers would be (0.5,1.5,2.75,4.25). The means learnt using modified K-palne regression approach are (0.495,1.495,2.745,4.25). This example demonstrates that our modified K-plane regression algorithm is robust to additive noise, and that it can learn discontinuous functions also well. This example also shows that the simple-minded K-plane regression performs poorly when there is noise in the training set.

Problem 2: In this problem the target function is a 3-piecewise affine function and we show the functions learnt by different approaches on noise-free as well as noisy training set. Figure 3 shows functions learnt using different approaches given the noise-free training examples. As can be seen, all algorithms learn a very good approximation of the target function (when given noise-free training data). The hinge hyperplanes method and SVR learn a continuous approximation while the K-plane and modified K-plane methods learn the discontinuous function.

We see that both K-plane and modified K-plane regression approach learn the function exactly. But MSE of K-plane method is much higher than modified K-plane method as can be seen in Table 3. The reason is as follows. In problem 2, the three sets (defining the partitioning of the domain of the function) corresponding to the three affine functions are [0,1), [1,2) and [2,3]. Moreover, $\forall x \in [0,1) \cup [2,3]$, $f(x) = x$. Thus the same affine function is assigned to two of the three disjoint sets. The K-plane regression approach tries to partition the training data so that for each partition we can learn a good function to fit the data; it does not care about whether the points in a partition are close together. Hence,

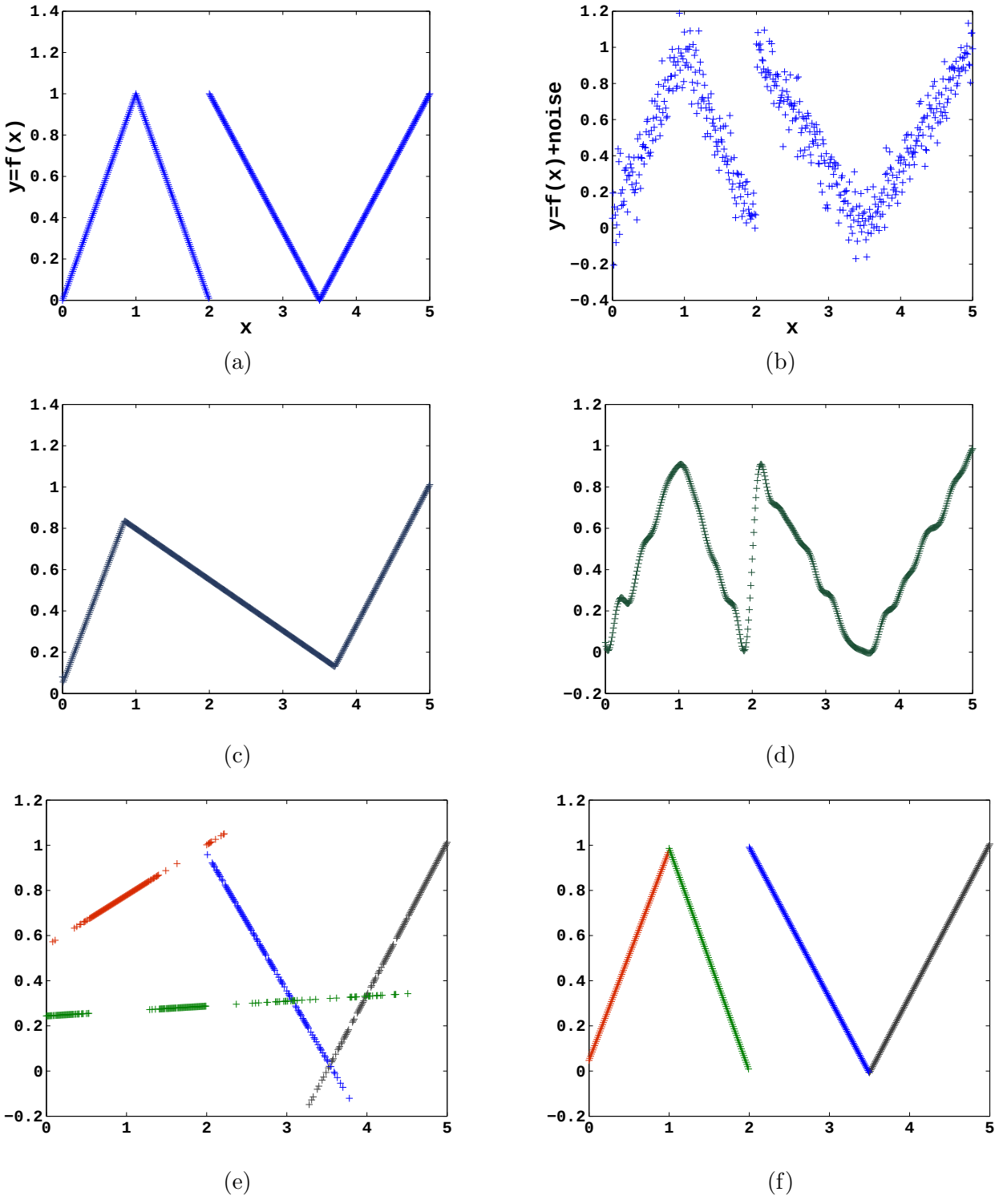

Figure 2: (a) 4-piecewise affine function f described in problem 1; (b) function f corrupted by additive Gaussian noise; functions learnt using (c) hinge hyperplane algorithm, (d) support vector regression, (e) K-plane regression approach and (f) modified K-plane regression approach, given noisy samples of f.

	Method	Parameters	MSE
Without Noise	K-plane	# hyperplanes = 3	0.7917
	Modified K-plane	# hyperplanes = 3	3.33×10^{-28}
	Hinge Hyperplane	# hinges = 13	0.008
	SVR	$C = 1024$, $\sigma = 16$, $\epsilon = 2^{-8}$	0.0041
With Noise	K-plane	# hyperplanes = 3	0.1352
	Modified K-plane	# hyperplanes = 3	0.011
	Hinge Hyperplane	# hinges = 23	0.0237
	SVR	$C = 1024$, $\sigma = 16$, $\epsilon = 2^{-8}$	0.0148

Table 3: MSE of different regression approaches on problem 2.

any partition of the the set $[0, 1) \cup [2, 3]$ into two parts (including the case where one part is null) would result in roughly the same value of the error function for the K-plane method. But, for prediction on a new point, we have to use the nearness of the new point to centriod of the partitions. Hence, if the partitions are bad then the final MSE can be very large. In this problem, when K-plane regression is given noise free samples, it always learnt only two hyperplanes irrespective of the value of K used (with the sets (S_k) corresponding to the remaining partitions being empty). The means of the two partions learnt were 1.505 and 1.4975. This clearly shows the algorithm has put $[0, 1) \cup [2, 3]$ into one partition. This leads to very poor prediction on test samples and high MSE in case of K-plane regression. In contrast, the means learnt using modified K-plane regression are 0.495, 1.5 and 2.505.

In the second part of this problem, we have added noise to the true function values in the training set as explained earlier. Figure 4 shows the functions learnt using different approaches given these noisy samples of the function. The MSE achieved by the learnt function on a test set under different algorithms are shown in Table 3. We see that only modified K-plane regression approach learns the target function exactly.

The function learnt by K-plane regression is very poor and its MSE is also high. This shows that, unlike in the earlier case, the K-plane regression algorithm could not even get the two affine functions correctly. Given the shape of function learnt on this problem by K-plane regression when the examples are noise-free, we can see that this algorithm is very sensitive to additive noise.

Both hinge hyperplanes method and SVR learn a good continuous approximation to the target function. However, these are not as good as the functions learnt by these algorithms on noise-free data of this problem. In contrast, the modified K-plane regression algorithm learns the discontinuous function exactly under our additive noise also. It also achieves the minimum MSE which is nothing but the noise variance as can be seen in Table 3.

Results on Real Datasets:

We now discuss performance of modified K-plane regression algorithm in comparison with other approaches on different real datasets. The results provided are based on 10- repetitions of 10-fold cross validation. We show average values and standard deviation of mean square error (MSE) and the time taken. The results are presented in Table 4-7.

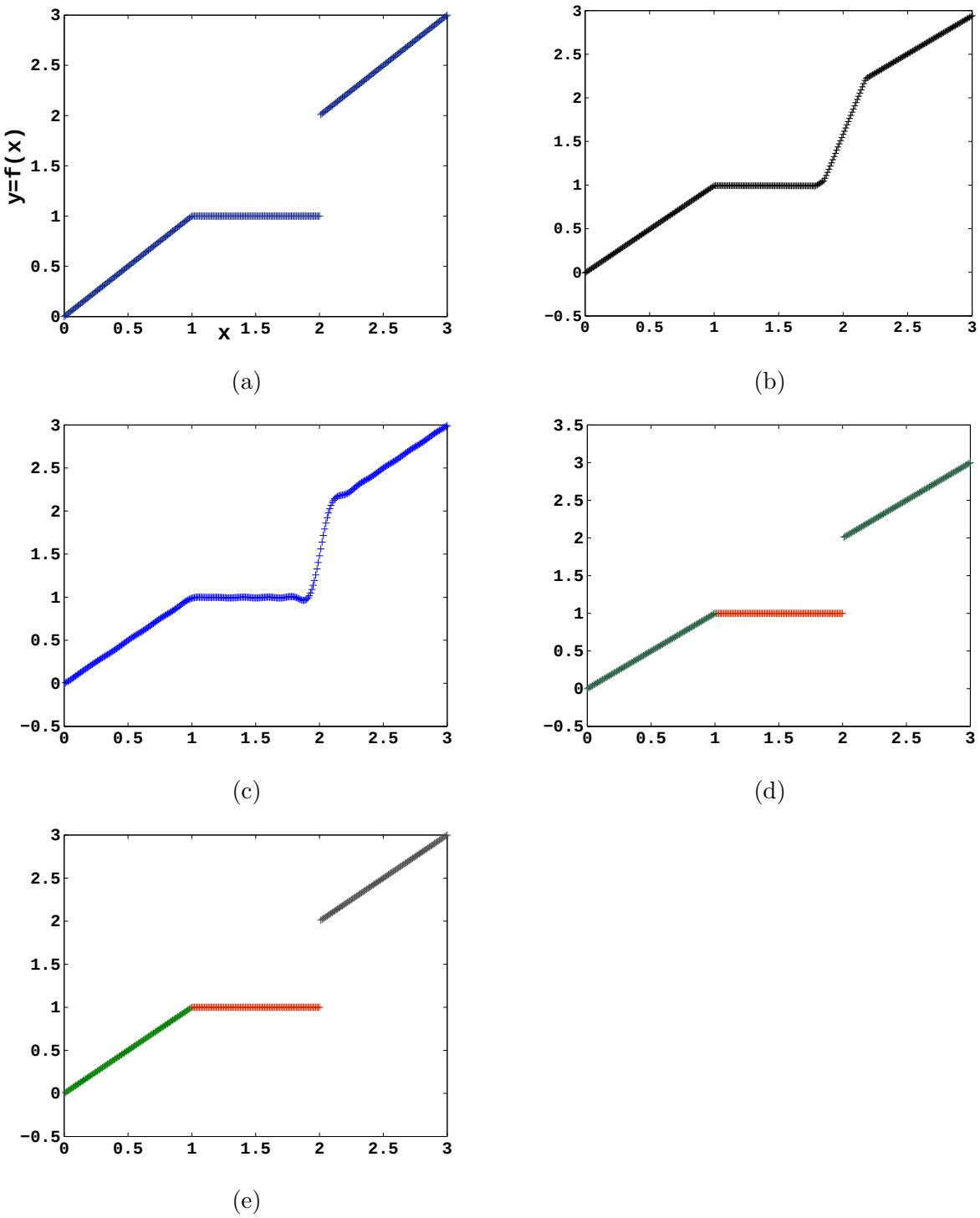

Figure 3: (a) 3-piecewise affine function f described in problem 2; functions learnt using (b) hinge hyperplane algorithm, (c) support vector regression, (d) K-plane regression approach and (e) modified K-plane regression approach, given noise-free samples of f.

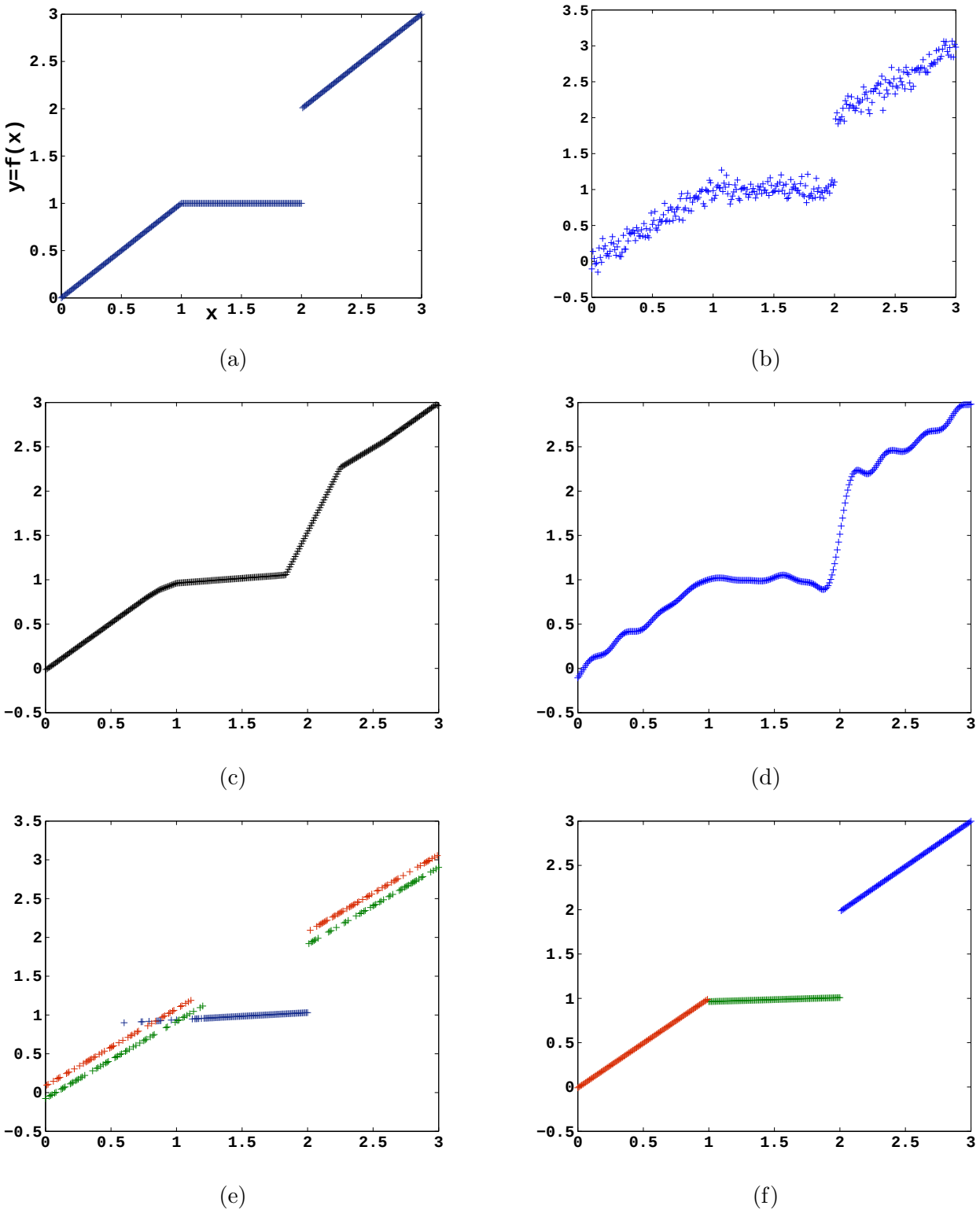

Figure 4: (a) 3-piecewise affine function f described in problem 2; (b) function f corrupted by additive Gaussian noise; functions learnt using (c) hinge hyperplane algorithm, (d) support vector regression, (e) K-plane regression approach and (f) modified K-plane regression approach, given noisy samples of f.

Method	Parameters	MSE	Time(sec)
K-plane	# hyperplanes = 2	17.15±0.85	0.01
	# hyperplanes = 3	27.47±2.74	0.03±0.003
	# hyperplanes = 4	30.29±1.77	0.04±0.005
	# hyperplanes = 5	39.67±9.11	0.06±0.012
Modified K-plane	# hyperplanes = 2	14.95±0.27	0.006
$\gamma = 100$	# hyperplanes = 3	14.72±0.53	0.01±0.001
	# hyperplanes = 4	14.25±0.62	0.014±0.001
	# hyperplanes = 5	13.92±0.78	0.02±0.002
Hinge Hyperplane	# hinges = 1	19.29±2.19	0.01±0.003
	# hinges = 2	16.45±1.34	0.04±0.006
	# hinges = 3	16.25±1.16	0.07±0.006
	# hinges = 4	16.06±0.98	0.11±0.008
	# hinges = 5	15.62±1.57	0.14±0.015
SVR	$C = 128$, $\sigma = 0.25$, $\epsilon = 2^{-8}$	10.08±0.42	0.17±0.01

Table 4: Comparison results of modified K-plane regression approach with other regression approaches on Housing Dataset.

Method	Parameters	MSE	Time(sec)
K-plane	# hyperplanes=2	10.44±0.17	0.11±0.02
	# hyperplanes=3	9.19±0.62	0.14±0.01
	# hyperplanes=4	9.87±1.45	0.27±0.03
	# hyperplanes=5	10.85±1.19	0.39±0.05
Modified K-plane	# hyperplanes=2	4.80±0.02	0.02±0.002
γ=100	# hyperplanes=3	4.68±0.03	0.04±0.005
	# hyperplanes=4	4.69±0.03	0.08±0.01
	# hyperplanes=5	4.68±0.03	0.08±0.01
Hinge Hyperplane	# hinges = 1	4.73±0.06	0.01±0.001
	# hinges = 2	4.53±0.03	0.08±0.01
	# hinges = 3	4.47±0.04	0.17±0.02
	# hinges = 4	4.41±0.02	0.28±0.02
	# hinges = 5	4.44±0.06	0.40±0.03
SVR	$C = 32$, $\sigma = 0.5$, $\epsilon = 0.5$	4.50±0.01	1.68±0.01

Table 5: Comparison results of modified K-plane regression approach with other regression approaches on Abalone Dataset.

Method	Parameters	MSE	Time(sec)
K-plane	# hyperplanes=2	10.34±0.22	0.02±0.001
	# hyperplanes=3	11.15±0.56	0.02±0.003
	# hyperplanes=4	13.08±1.10	0.04±0.003
	# hyperplanes=5	13.72±0.77	0.05±0.003
Modified K-plane	# hyperplanes=2	8.55±0.11	0.006
γ=100	# hyperplanes=3	8.72±0.25	0.01±0.003
	# hyperplanes=4	8.82±0.75	0.01±0.001
	# hyperplanes=5	8.83±0.69	0.01±0.002
Hinge Hyperplane	# hinges = 1	9.81±0.52	0.003
	# hinges = 2	9.03±0.53	0.02±0.002
	# hinges = 3	8.75±0.37	0.03±0.01
	# hinges = 4	8.58±0.35	0.05±0.009
	# hinges = 5	8.35±0.39	0.08±0.005
SVR	C=16, $\sigma = 1$, $\epsilon = 0.25$	6.80±0.26	0.03

Table 6: Comparison results of modified K-plane regression approach with other regression approaches on Auto-mpg Dataset.

Method	Parameters	MSE	Time(sec)
K-plane	# hyperplanes=2	61.81±7.17	0.39±0.05
	# hyperplanes=3	19.48±0.49	0.48±0.07
	# hyperplanes=4	15.88±0.92	0.92±0.13
	# hyperplanes=5	19.98±1.03	1.19±0.20
Modified K-plane	# hyperplanes=2	154.45±12.61	0.15±0.02
$\gamma = 100$	# hyperplanes=3	23.47±1.56	0.24±0.03
	# hyperplanes=4	11.88±0.15	0.48±0.04
	# hyperplanes=5	11.80±0.24	0.66±0.10
	# hyperplanes=6	10.98±0.14	0.70±0.16
Hinge Hyperplane	# hinges = 1	29.40±21.57	0.05±0.002
	# hinges = 2	11.39±0.32	0.17±0.017
	# hinges = 3	10.77±0.27	0.38±0.047
	# hinges = 4	10.66±0.33	0.64±0.062
	# hinges = 5	10.06±0.13	0.98±0.055
SVR	$C = 256$, $\sigma = 1$, $\epsilon = 0.5$	8.47±0.11	23.16±0.31

Table 7: Comparison results of modified K-plane regression approach with other regression approaches on Computer Activity Dataset.

We see that for all datasets, the MSE achieved by the simple K-plane regression method is highest among all algorithms. Thus, though the K-plane regression method is conceptually simple and appealing, its performance is not very good.

The modified K-plane regression algorithm performs much better than K-plane regression not only in terms of MSE but also in terms of time taken. The reason why modified K-planes method takes lesser time is that it converges in fewer iterations. This happens because modified K-plane regression algorithm gives importance to the connectedness of the clusters also. As a result, number of transitions of points from one cluster to another after every iteration are lesser and thus the clusters stabilize after fewer iterations.

The performamnce of modified K-plane regression algorithm is comparable to that of hinge hyperplane algorithm in terms of MSE. It performs better than hinge hyperplane method on Housing dataset. On Auto-Mpg dataset, Abalone dataset and Computer Activity dataset, the minimum MSE of modified K-plane regression approach is only a little higher than the minimum MSE of hinge hyperplane method. Modified K-plane regression algorithm is also faster than hinge hyperplane method on all data sets.

On all problems except on the Abolone dataset, the SVR algorithms achieves better MSE than modified K-plane regression algorithm. However, we observe that modified K-plane regression is significantly faster than SVR. In SVR, the complexity of dual optimization problem is $O(N^3)$, where N is the number of points. In contrast, in modified K-plane regression, at each iteration, the major computation is finding K linear regression functions. The time complexity of each iteration in modified K-plane regression is $O(K(d+1)^3)$ which is very less than $O(N^3)$ if $N >> d$.

Thus, we see that overall, modified K-plane regression is a very attractive method for learning nonlinear regression functions by approximating them as piecewise linear functions. It is conceptually simple and the algorithm is very efficient. Its performance is comparable to that of SVR or hinge hyperplanes method in terms of accuracy. It is significantly faster than SVR and is also faster than hinge hyperplane method. Further, unlike all other current regression function learning algorithms, this method is capable of learning discontinuous functions also.

6. Conclusions

In this paper, we considered the problem of learning piecewise linear regression models. We proposed an interesting and simple algorithm to learn such functions. The proposed method is capable of learning discontinuous functions also. Through simulation experiments we showed that the performance of the proposed method is good and is comparable to state-of-art in regression function learning.

The basic idea behind the proposed method is very simple. Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be the training dataset. We essentially want to find a way to partition the set $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ into K sets such that we can find a good linear fit for the targets (i.e., y_i) of points in each partition. The algorithm achieves this by repeatedly partitioning the points and fitting linear models. After each model fit, we repartition the points based on the closeness of targets to the current models. We called this the K-plane regression algorithm. This algorithm is conceptually simple and is similar in spirit to the K-means clustering method. While such

an idea has been discussed in different contexts, we have not come across this algorithm proposed and empirically investigated for nonlinear regression.

Though this idea is interesting, as we showed here, it has several drawbacks. As the results in previous section show, this algorithm performs poorly even on one dimensional problems.

In this paper we have also proposed a modification of the above method which performs well as a regression learning method. In our modified K-plane regression algorithm, during the process of repeatedly partitioning feature vectors and fitting linear models, we make the partions so that we get good linear models and, also, the points of a partition are all close together. This idea is easily incorporated into the algorithm by expanding the parameter vector to be learnt and by modifying the objective function to be minimized. The resulting algorithm essentially does one step of linear regression and one step of K-means clustering in each iteration.

Through empirical studies, we showed that the modified K-plane regression algorithm is very effective. Its performance on some real data sets is comparable to that of nonlinear SVR in terms of accuracy while the proposed method is much faster than SVR. The proposed method is better than the hinge hyperplane algorithm which is arguably the best method today for learning piecewise linear functions. Through two synthetic one-dimensional problems, we also showed that the proposed method has better robustness to additive noise than the other methods and that it is capable of learning discontinuous functions also.

We feel that the proposed method opens up interesting possibilities of designing algorithms for learning piecewise linear functions. As mentioned earlier, simultaneous estimation of optimal partitions and optimal models for each partition is computationally intractable. Hence an interesting and difficult open question is to establish theoretical bounds on the performance of the modified K-plane regression method. While we showed that the method can be viewed as a limiting case EM algorithm under reasonable probability model, a lot of work needs to be done to understand how close to optimum can such methods converge to.

References

D.J. Newman A. Asuncion. *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences, 2007. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

E. Amaldi and M. Mattavelli. The MIN PFS Problem and Piecewise Linear Model Estimation. *Discrete Applied Mathematics*, 118(1-2):115–143, April 2002.

Alberto Bemporad, Andrea Garulli, Simone Paoletti, and Antonio Vicino. A Greedy Approach to Identification of Piecewise Affine Models. In *Proceedings of the 6th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 97–112, Prague, Czech Republic, April 2003.

Alberto Bemporad, Andrea Garulli, Simone Paoletti, and Antonio Vicino. A Bounded Error Approach to Piecewise Affine System Identification. *IEEE Transaction on Automatic Control*, 50(10):1567–1580, October 2005.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, first edition, 2006.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth and Brooks, Belmont, California, U.S.A., 1984.

Leo Breiman. Hinging Hyperplane for Regression, Classification and Function Approximation. *IEEE Transaction on Information Theory*, 39(3):999–1013, May 1993.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM : A Library for Support Vector Machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Elements of Statistical Learning Theory*. Springer, 2001.

R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, March 1991.

A. K. Deb Jayadeva and S. Chandra. Algorithm for Building a Neural Network for Function Approximation. *IEE Proc.-Circuits Devices Systems*, 149(516):301–307, October-December 2002.

Michael I. Jordan and Robert A. Jacobs. Hierarchical Mixture of Experts and EM Algorithm. *Neural Computation*, 6(2):181–214, March 1994.

Alessandro Magnani and Stephen P. Boyd. Convex piecewise linear fitting. *Optimization and Engineering*, 10(1):1–17, March 2009.

S. Paoletti, A. L. Juloski, Ferrari-Trecate, and R. Vidal. Identification of Hybrid Systems: A Tutorial. *European Journal of Control*, 13(2/3):242–260, 2007.

P. Pucar and J. Sjöberg. On the Hinge-Finding Algorithm for Hinging Hyperplanes. *IEEE Transaction on Information Theory*, 44(3):1310–1319, May 1998.

Alex J. Smola and Bernhard Schölkopf. A Tutorial on Support Vector Regression. NeuroCOLT2 Technical Report Series NC2-TR-1998-030, GMD, October 1998.

Shuning Wang and Xusheng Sun. Generalization of Hinging Hyperplanes. *IEEE Transaction on Information Theory*, 51(12):4425–4431, December 2005.

Steven Richard Waterhouse. *Classification and Regression using Mixtures of Experts*. PhD thesis, Department of Engineering, University of Cambridge, 1997.

L. Xu, M. I. Jordan, and G. E. Hinton. An alternative model for mixtures of experts. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 633–640, Denver, CO, USA, November 1995.